The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV012-SW-42

VxWorks Device Driver

32 differential I/O Lines with Interrupts

Version 2.0.x

User Manual

Issue 2.0.0

March 2014

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7 25469 Halstenbek, Germany
Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com www.tews.com

TDRV012-SW-42

VxWorks Device Driver

32 differential I/O Lines with Interrupts

Supported Modules:
TPMC683

Issue	Description	Date
1.0.0	First Issue	November 25, 2008
2.0.0	First Issue	March 25, 2014

Table of Contents

1 Introduction

1.1 Device Driver

The TDRV012-SW-42 VxWorks device driver software allows the operation of the supported PMC conforming to the VxWorks I/O system specification.

The TDRV012-SW-42 release contains independent driver sources for the old legacy (pre-VxBus) and the new VxBus-enabled driver model. The VxBus-enabled driver is recommended for new developments with later VxWorks 6.x release and mandatory for VxWorks 64-bit and SMP systems.

Both drivers, legacy and VxBus, share the same application programming interface (API) and device-independent basic I/O interface with open(), close() and ioctl() functions. The basic I/O interface is only for backward compatibility with existing applications and should not be used for new developments.

The TDRV012-SW-42 device driver supports the following features:

- Configure input/output direction of each line
- read state of input lines
- write to output lines
- wait for interrupt events (rising/falling edge) on each input line

The TDRV012-SW-42 supports the modules listed below:

TPMC683	32 differential I/O Lines with Interrupts	(PMC)

In this document all supported modules and devices will be called TDRV012. Specials for certain devices will be advised.

To get more information about the features and use of supported devices it is recommended to read the manuals listed below.

TPMC683 User Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV012-SW-42':

TDRV012-SW-42-2.0.0.pdf	PDF copy of this manual
TDRV012-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TDRV012-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TDRV012-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tdrv012':

tdrv012drv.c	TDRV012 device driver source
tdrv012def.h	TDRV012 driver include file
tdrv012.h	TDRV012 include file for driver and application
tdrv012api.c	TDRV012 API file
Makefile	Driver Makefile
40tdrv012.cdf	Component description file for VxWorks development tools
tdrv012.dc	Configuration stub file for direct BSP builds
tdrv012.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tdrv012exa.c	Example application

The archive TDRV012-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tdrv012':

tdrv012drv.c	TDRV012 device driver source
tdrv012def.h	TDRV012 driver include file
tdrv012.h	TDRV012 include file for driver and application
tdrv012pci.c	TDRV012 device driver source for x86 based systems
tdrv012api.c	TDRV012 API file
tdrv012exa.c	Example application
include/tdhal.h	Hardware dependent interface functions and definitions

2.1 Legacy vs. VxBus Driver

In later VxWorks 6.x releases, the old VxWorks 5.x legacy device driver model was replaced by VxBus-enabled device drivers. Legacy device drivers are tightly coupled with the BSP and the board hardware. The VxBus infrastructure hides all BSP and hardware differences under a well defined interface, which improves the portability and reduces the configuration effort. A further advantage is the improved performance of API calls by using the method interface and bypassing the VxWorks basic I/O interface.

VxBus-enabled device drivers are the preferred driver interface for new developments.

The checklist below will help you to make a decision which driver model is suitable and possible for your application:

Legacy Driver	VxBus Driver
▪ VxWorks 5.x releases ▪ VxWorks 6.5 and earlier releases ▪ VxWorks 6.x releases without VxBus PCI bus support	▪ VxWorks 6.6 and later releases with VxBus PCI bus ▪ SMP systems (only the VxBus driver is SMP safe) ▪ 64-bit systems (only the VxBus driver is 64-bit compatible)

TEWS TECHNOLOGIES recommends not using the VxBus Driver before VxWorks release 6.6. In previous releases required header files are missing and the support for 3rd-party drivers may not be available.

2.2 VxBus Driver Installation

Because Wind River doesn't provide a standard installation method for 3rd party VxBus device drivers the installation procedure needs to be done manually.

In order to perform a manual installation extract all files from the archive TDRV012-SW-42-VXBUS.zip to the typical 3rd party directory *installDir/vxworks-6.x/target/3rdparty* (whereas *installDir* must be substituted by the VxWorks installation directory).

After successful installation the TDRV012 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tdrv012.*

At this point the TDRV012 driver is not configurable and cannot be included with the kernel configuration tool in a Wind River Workbench project. To make the driver configurable the driver library for the desired processor (CPU) and supported build tool (TOOL) must be built in the following way:

(1) Open a VxWorks development shell (e.g. C:\WindRiver\wrenv.exe -p vxworks-6.7)

(2) Change into the driver installation directory
installDir/vxworks-6.x/target/3rdparty/tews/tdrv012

(3) Invoke the build command for the required processor and build tool with optional VXBUILD argument
make CPU=cpuName TOOL=tool [VXBUILD=xxx]

For Windows hosts this may look like this:

```
> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv012
> make CPU=PENTIUM4 TOOL=diab
```

To build SMP-enabled libraries, the argument VXBUILD=SMP must be added to the command line

```
> make CPU=PENTIUM4 TOOL=diab VXBUILD=SMP
```

To build 64-bit libraries, the argument VXBUILD=LP64 must be added to the command line

```
> make TOOL=gnu CPU=CORE VXBUILD=LP64
```

For 64-bit SMP-enabled libraries a build command may look like this

```
> make TOOL=gnu CPU=CORE VXBUILD="LP64 SMP"
```

To integrate the TDRV012 driver with the VxWorks development tools (Workbench), the component configuration file *40tdrv012.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv012
C:> copy 40tdrv000.cdf \Windriver\vxworks-6.7\target\config\comps\vxWorks
```

In VxWorks 6.7 and newer releases the kernel configuration tool scans the CDF file automatically and updates the *CxrCat.txt* cache file to provide component parameter information for the kernel configuration tool as long as the timestamp of the copied CDF file is newer than the one of the *CxrCat.txt*. If your copy command preserves the timestamp, force to update the timestamp by a utility, such as *touch*.

In earlier VxWorks releases the CxrCat.txt file may not be updated automatically. In this case, remove or rename the original *CxrCat.txt* file and invoke the make command to force recreation of this file.

```
C:> cd \Windriver\vxworks-6.7\target\config\comps\vxWorks
C:> del CxrCat.txt
C:> make
```

After successful completion of all steps above and restart of the Wind River Workbench, the TDRV012 driver and API can be included in VxWorks projects by selecting the *"TEWS TDRV012 Driver" and "TEWS TDRV012 API"* components in the *"hardware (default) - Device Drivers"* folder with the kernel configuration tool.

2.2.1 Direct BSP Builds

In development scenarios with the direct BSP build method without using the Workbench or the vxprj command-line utility, the TDRV012 configuration stub files must be copied to the directory *installDir/vxworks-6.x/target/config/comps/src/hwif.* Afterwards the *vxbUsrCmdLine.c* file must be updated by invoking the appropriate make command.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv012
C:> copy tdrv012.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tdrv012.dr \Windriver\vxworks-6.7\target\config\comps\src\hwif

C:> cd \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> make vxbUsrCmdLine.c
```

2.3 Legacy Driver Installation

2.3.1 Include Device Driver in VxWorks Projects

For including the TDRV012-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Extract all files from the archive TDRV012-SW-42-LEGACY.zip to your project directory.

(2) Add the device drivers C-files to your project.
Make a right click to your project in the 'Workspace' window and use the 'Add Files ...' topic.
A file select box appears, and the driver files in the tdrv012 directory can be selected.

(3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)

2.3.2 Special Installation for Intel x86 based Targets

The TDRV012 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro **CPU_FAMILY**. If the content of this macro is equal to *I80X86* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, the required device memory spaces can't be accessed.

To solve this problem a MMU mapping entry has to be added for the required TDRV012 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tdrv012pci.c** contains the function *tdrv012PciInit().* This routine finds out all TDRV012 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*).

The right place to call the function *tdrv012PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window):

```
tdrv012PciInit();
```

Be sure that the function is called prior to MMU initialization otherwise the TDRV012 PCI spaces remains unmapped and an access fault occurs during driver initialization.

Modifying the sysLib.c file will change the sysLib.c in the BSP path. Remember this for future projects and recompilations.

2.3.3 BSP Dependent Adjustments

The driver includes a file called *include/tdhal.h* which contains functions and definitions for BSP adaptation. It may be necessary to modify them for BSP specific settings. Most settings can be made automatically by conditional compilation set by the BSP header files, but some settings must be configured manually. There are two way of modification, first you can change the *include/tdhal.h* and define the corresponding definition and its value, or you can do it, using the command line option *–D*.

There are 3 offset definitions (*USERDEFINED_MEM_OFFSET*, *USERDEFINED_IO_OFFSET*, and *USERDEFINED_LEV2VEC*) that must be configured if a corresponding warning message appears during compilation. These definitions always need values. Definition values can be assigned by command line option *-D<definition>=<value>*.

Definition	Description
USERDEFINED_MEM_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI memory space access
USERDEFINED_IO_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI I/O space access
USERDEFINED_LEV2VEC	The value of this definition must be set to the difference of the interrupt vector (used to connect the ISR) and the interrupt level (stored to the PCI header)

Another definition allows a simple adaptation for BSPs that utilize a *pciIntConnect()* function to connect shared (PCI) interrupts. If this function is defined in the used BSP, the definition of *USERDEFINED_SEL_PCIINTCONNECT* should be enabled. The definition by command line option is made by *-D<definition>.*

Please refer to the BSP documentation and header files to get information about the interrupt connection function and the required offset values.

2.3.4 System Resource Requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 1 KB
Stack	< 1 KB	---
Semaphores	---	32

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

3 API Documentation

3.1 General Functions

3.1.1 tdrv012Open

NAME

tdrv012Open – Open a Device

SYNOPSIS

```
TDRV012_HANDLE tdrv012Open
(
      char                              *DeviceName
)
```

DESCRIPTION

Before I/O can be performed to a device, a file descriptor must be opened by a call to this function.

PARAMETERS

DeviceName

This parameter points to a null-terminated string that specifies the name of the device. The first TDRV012 device is named /tdrv012/0, the second /tdrv012/1, and so on.

EXAMPLE

```
#include “tdrv012api.h”

TDRV012_HANDLE hdl;

/*
** open file descriptor to device
*/
hdl = tdrv012Open(“/tdrv012/0”);
if (hdl == NULL)
{
     /* handle open error */
}
```

RETURNS

A device handle, or NULL if the function fails. An error code will be stored in *errno*.

ERROR CODES

All error codes are standard error codes set by the I/O system.

3.1.2 tdrv012Close

NAME

tdrv012Close – Close a Device

SYNOPSIS

```
TDRV012_STATUS tdrv012Close
(
    TDRV012_HANDLE          hdl
)
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv012api.h”

TDRV012_HANDLE     hdl;
TDRV012_STATUS     result;

/*
** close file descriptor to device
*/
result = tdrv012Close( hdl );
if (result != TDRV012_OK)
{
    /* handle close error */
}
```

RETURNS

On success TDRV012_OK, or an appropriate error code.

ERROR CODES

Error Code	Description
TDRV012_ERR_INVALID_HANDLE	The specified TDRV012_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2 Device Access Functions

3.2.1 tdrv012Read

NAME

tdrv012Read – Read current I/O Value

SYNOPSIS

```
TDRV012_STATUS tdrv012Read
(
    TDRV012_HANDLE          hdl,
    unsigned int            *pIoValue
)
```

DESCRIPTION

This function reads the current state of the input and output lines of the specified device.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pIoValue

This value is a pointer to a uint32_t 32bit data buffer which receives the current I/O value. Both input and output values are returned. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

EXAMPLE

```
#include “tdrv012api.h”

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;
unsigned int    IoValue;

/*
** read current I/O value
*/
result = tdrv012Read( hdl, &IoValue );
if (result == TDRV012_OK)
{
     printf( “I/O Value: 0x%08X\n”, IoValue );
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV012_ERR_INVALID_HANDLE	The specified TDRV012_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.2 tdrv012WriteMask

NAME

tdrv012WriteMask – Write relevant Bits of Output Value

SYNOPSIS

```
TDRV012_STATUS tdrv012WriteMask
(
    TDRV012_HANDLE      hdl,
    unsigned int        OutputValue,
    unsigned int        BitMask
);
```

DESCRIPTION

This function writes relevant bits of a new output value for the specified device.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

OutputValue

This value specifies the new output value. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on.

BitMask

This parameter specifies the bitmask. Only active bits (1) will be written to the output register, all other output lines will be left unchanged. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on.

EXAMPLE

```
#include “tdrv012api.h”

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;

/*
** write new output value:
** set 2nd  (bit 1) output line to ON, and 7th (bit 6) output line to OFF.
** leave all other output lines unchanged.
*/
result = tdrv012WriteMask(
              hdl,
              (1 << 1),
              (1 << 1) | (1 << 6)
         );
if (result == TDRV012_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV012_ERR_INVALID_HANDLE	The specified TDRV012_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.3 tdrv012OutputSet

NAME

tdrv012OutputSet – Set single Output Lines to ON

SYNOPSIS

```
TDRV012_STATUS tdrv012OutputSet
(
    TDRV012_HANDLE      hdl,
    unsigned int        OutputValue
)
```

DESCRIPTION

This function sets single output lines to ON leaving other output lines in the current state.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

OutputValue

This value specifies the new output value. Active (1) bits will set the corresponding output line to ON, unset (0) bits will not have an effect on the corresponding output lines. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on.

EXAMPLE

```
#include “tdrv012api.h”

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;

/*
** write new output value:
** set 2nd  (bit 1) and 3rd (bit 2) output line to ON.
** leave all other output lines unchanged.
*/
result = tdrv012OutputSet(
              hdl,
              (1 << 1) | (1 << 2)
         );
if (result == TDRV012_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV012_ERR_INVALID_HANDLE	The specified TDRV012_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.4 tdrv012OutputClear

NAME

tdrv012OutputClear – Clear single Output Lines to OFF

SYNOPSIS

```
TDRV012_STATUS tdrv012OutputClear
(
    TDRV012_HANDLE      hdl,
    unsigned int        OutputValue
)
```

DESCRIPTION

This function clears single output lines to OFF leaving other output lines in the current state.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

OutputValue

This value specifies the new output value. Active (1) bits will clear the corresponding output line to OFF, unset (0) bits will not have an effect on the corresponding output lines. Bit 0 of this value corresponds to the first output line, bit 1 corresponds to the second output line and so on.

EXAMPLE

```
#include “tdrv012api.h”

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;

/*
** write new output value:
** clear 2nd  (bit 1) and 4th (bit 3) output line to OFF.
** leave all other output lines unchanged.
*/
result = tdrv012OutputClear(
              hdl,
              (1 << 1) | (1 << 3)
         );
if (result == TDRV012_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV012_ERR_INVALID_HANDLE	The specified TDRV012_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.5 tdrv012ConfigureDirection

NAME

tdrv012ConfigureDirection – Configure Input/Output Direction of I/O Lines

SYNOPSIS

```
TDRV012_STATUS tdrv012ConfigureDirection
(
    TDRV012_HANDLE      hdl,
    unsigned int        DirectionValue,
    unsigned int        DirectionMask
)
```

DESCRIPTION

This function configures the direction (input/output) of specific I/O lines. Only specific lines specified by a mask are affected.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

DirectionValue

This value specifies the direction of the corresponding I/O lines. An active (1) bit will configure the corresponding I/O line to OUTPUT, an unset (0) bit will configure the corresponding I/O line to INPUT. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

DirectionMask

This parameter specifies the bitmask. Only active bits (1) will have an effect on the I/O direction, the direction of all other I/O lines will be left unchanged. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

EXAMPLE

```
#include “tdrv012api.h”

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;

/*
** configure new I/O direction:
** set lowest 8 I/O lines to OUTPUT, and highest 8 I/O lines to INPUT.
** leave all other I/O lines unchanged.
*/
result = tdrv012ConfigureDirection(
              hdl,
              (0x00 << 24) | (0xff << 0),
              (0xff << 24) | (0xff << 0)
         );
if (result == TDRV012_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV012_ERR_INVALID_HANDLE	The specified TDRV012_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.6 tdrv012ReadDirection

NAME

tdrv012ReadDirection – Read current Input/Output Direction Configuration of I/O Lines

SYNOPSIS

```
TDRV012_STATUS tdrv012ReadDirection
(
    TDRV012_HANDLE      hdl,
    unsigned int        *pDirectionValue
)
```

DESCRIPTION

This function reads the current direction configuration (input/output) of the I/O lines.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pDirectionValue

This value is a pointer to an *unsigned int* 32bit data buffer which receives the current I/O direction configuration. Active (1) bits represent OUTPUT lines, unset (0) bits represent INPUT lines. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

EXAMPLE

```
#include “tdrv012api.h”

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;
unsigned int    DirectionValue;

/*
** read current I/O direction configuration
*/
result = tdrv012ReadDirection(
              hdl,
              &DirectionValue
         );
if (result == TDRV012_OK)
{
     printf(“Current direction configuration (1=OUTPUT, 0=INPUT):\n”);
     printf(“  0x%08X\n”, DirectionValue);
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV012_ERR_INVALID_HANDLE	The specified TDRV012_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.7 tdrv012WaitEvent

NAME

tdrv012WaitEvent – Wait for specific Transitions on I/O Lines

SYNOPSIS

```
TDRV012_STATUS tdrv012WaitEvent
(
    TDRV012_HANDLE      hdl,
    unsigned int        mask_high,
    unsigned int        mask_low,
    int                 timeout,
    unsigned int        *pIoValue,
    unsigned int        *pStatusHigh,
    unsigned int        *pStatusLow
);
```

DESCRIPTION

This function blocks until at least one of the specified events or a timeout occurs.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

mask_high

This parameter specifies on which input line a HIGH transition should occur to trigger an event. Multiple input lines may be specified. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

mask_low

This parameter specifies on which input line a LOW transition should occur to trigger an event. Multiple input lines may be specified. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

timeout

This parameter specifies the time the function should wait for the event. The timeout is specified in milliseconds. Use -1 to wait indefinitely for the event.

pIoValue

This value is a pointer to an *unsigned int* 32bit data buffer which returns the state of the input lines at the moment the event is served by the interrupt service routine. Keep in mind that there is a system-dependent interrupt latency, so it is not guaranteed that this value is the actual input state at the time of the event.

pStatusHigh

This parameter is a pointer to an *unsigned int* 32bit data buffer which returns on which input lines a HIGH transition has occurred for the current wait job. This parameter is a bitmask, where bit 0 corresponds to I/O line 0, bit 1 corresponds to I/O line 1 and so on.

pStatusLow

This parameter is a pointer to an *unsigned int* 32bit data buffer which returns on which input lines a LOW transition has occurred for the current wait job. This parameter is a bitmask, where bit 0 corresponds to I/O line 0, bit 1 corresponds to I/O line 1 and so on.

EXAMPLE

```
#include “tdrv012api.h”

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;
unsigned int    IoValue, StatusHigh, StatusLow;

/*
** wait at least 1000ms for events:
** HIGH transition on I/O line 0 or
** LOW transition on I/O line 1 or
** HIGH/LOW=ANY transition on I/O line 2
*/
result = tdrv012WaitEvent(
              hdl,
              (1 << 2) | (1 << 0),
              (1 << 2) | (1 << 1),
              1000,
              &IoValue,
              &StatusHigh,
              &StatusLow
         );

…
```

…

```
if (result == TDRV012_OK)
{
     printf("  Current I/O status    : 0x%08X\n", IoValue);
     printf("  HIGH transition status: 0x%08X\n", StatusHigh);
     printf("  LOW  transition status: 0x%08X\n", StatusLow);
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV012_ERR_INVALID_HANDLE	The specified TDRV012_HANDLE is invalid.
TDRV012_ERR_BUSY	Too many concurrent wait jobs pending (max. 100)
TDRV012_ERR_TIMEOUT	Timeout. None of the specified events occurred.

Other returned error codes are system error conditions.

3.2.8 tdrv012WaitHigh

NAME

tdrv012WaitHigh – Wait for HIGH Transitions on specific I/O Lines

SYNOPSIS

```
TDRV012_STATUS tdrv012WaitHigh
(
    TDRV012_HANDLE      hdl,
    unsigned int        mask,
    int                 timeout,
    unsigned int        *pIoValue,
    unsigned int        *pStatus
);
```

DESCRIPTION

This function blocks until at least one of the specified HIGH events or a timeout occurs.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

mask

This parameter specifies on which input line the HIGH transition should occur to trigger an event. Multiple input lines may be specified. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

timeout

This parameter specifies the time the function should wait for the event. The timeout is specified in milliseconds. Use -1 to wait indefinitely for the event.

pIoValue

This value is a pointer to an *unsigned int* 32bit data buffer which returns the state of the input lines at the moment the event is served by the interrupt service routine. Keep in mind that there is a system-dependent interrupt latency, so it is not guaranteed that this value is the actual input state at the event.

pStatus

This parameter is a pointer to an *unsigned int* 32bit data buffer which returns on which input lines a HIGH transition has occurred for the current wait job. This parameter is a bitmask, where bit 0 corresponds to I/O line 0, bit 1 corresponds to I/O line 1 and so on.

EXAMPLE

```
#include “tdrv012api.h”

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;
unsigned int       IoValue;
unsigned int       Status;

/*
** wait at least 1000ms for HIGH transition events:
** HIGH transition on I/O line 31
*/
result = tdrv012WaitHigh(
              hdl,
              (1 << 31),
              1000,
              &IoValue,
              &Status
         );
if (result == TDRV012_OK)
{
     printf(“  Current I/O status    : 0x%08X\n”, IoValue);
     printf(“  HIGH transition status: 0x%08X\n”, Status);
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV012_ERR_INVALID_HANDLE	The specified TDRV012_HANDLE is invalid.
TDRV012_ERR_BUSY	Too many concurrent wait jobs pending (max. 100)
TDRV012_ERR_TIMEOUT	Timeout. None of the specified events occurred.

Other returned error codes are system error conditions.

3.2.9 tdrv012WaitLow

NAME

tdrv012WaitLow – Wait for LOW Transitions on specific I/O Lines

SYNOPSIS

```
TDRV012_STATUS tdrv012WaitLow
(
    TDRV012_HANDLE      hdl,
    unsigned int        mask,
    int                 timeout,
    unsigned int        *pIoValue,
    unsigned int        *pStatus
)
```

DESCRIPTION

This function blocks until at least one of the specified LOW events or a timeout occurs.

PARAMETERS

handle

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

mask

This parameter specifies on which input line the LOW transition should occur to trigger an event. Multiple input lines may be specified. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

timeout

This parameter specifies the time the function should wait for the event. The timeout is specified in milliseconds. Use -1 to wait indefinitely for the event.

pIoValue

This value is a pointer to an *unsigned int* 32bit data buffer which returns the state of the input lines at the moment the event is served by the interrupt service routine. Keep in mind that there is a system-dependent interrupt latency, so it is not guaranteed that this value is the actual input state at the event.

pStatus

This parameter is a pointer to an *unsigned int* 32bit data buffer which returns on which input lines a LOW transition has occurred for the current wait job. This parameter is a bitmask, where bit 0 corresponds to I/O line 0, bit 1 corresponds to I/O line 1 and so on.

EXAMPLE

```
#include “tdrv012api.h”

TDRV012_HANDLE  hdl;
TDRV012_STATUS  result;
unsigned int    IoValue;
unsigned int    Status;

/*
** wait at least 1000ms for LOW transition events:
** LOW transition on I/O line 31
*/
result = tdrv012WaitLow(
              hdl,
              (1 << 31),
              1000,
              &IoValue,
              &Status
         );
if (result == TDRV012_OK)
{
     printf(“  Current I/O status   : 0x%08X\n”, IoValue);
     printf(“  LOW transition status: 0x%08X\n”, Status);
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV012_ERR_INVALID_HANDLE	The specified TDRV012_HANDLE is invalid.
TDRV012_ERR_BUSY	Too many concurrent wait jobs pending (max. 100)
TDRV012_ERR_TIMEOUT	Timeout. None of the specified events occurred.

Other returned error codes are system error conditions.

3.2.10 tdrv012WaitAny

NAME

tdrv012WaitAny – Wait for HIGH or LOW Transitions on specific I/O Lines

SYNOPSIS

```
TDRV012_STATUS tdrv012WaitAny
(
    TDRV012_HANDLE      hdl,
    unsigned int        mask,
    int                 timeout,
    unsigned int        *pIoValue,
    unsigned int        *pStatus
)
```

DESCRIPTION

This function blocks until at least one of the specified HIGH or LOW events or a timeout occurs.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

mask

This parameter specifies on which input line the HIGH or LOW transition should occur to trigger an event. Multiple input lines may be specified. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

timeout

This parameter specifies the time the function should wait for the event. The timeout is specified in milliseconds, although the granularity is in seconds. Use -1 to wait indefinitely for the event.

pIoValue

This value is a pointer to an *unsigned int* 32bit data buffer which returns the state of the input lines at the moment the event is served by the interrupt service routine. Keep in mind that there is a system-dependent interrupt latency, so it is not guaranteed that this value is the actual input state at the event.

pStatus

This parameter is a pointer to an *unsigned int* 32bit data buffer which returns on which input lines a HIGH or LOW transition has occurred for the current wait job. This parameter is a bitmask, where bit 0 corresponds to I/O line 0, bit 1 corresponds to I/O line 1 and so on. It is not possible to distinguish between a HIGH or LOW event. To do this, use tdrv012waitEvent() instead.

EXAMPLE

```
#include “tdrv012api.h”

TDRV012_HANDLE     hdl;
TDRV012_STATUS     result;
unsigned int       IoValue;
unsigned int       Status;

/*
** wait at least 1000ms for HIGH or LOW transition events:
** any transition on I/O line 0
*/
result = tdrv012WaitAny(
              hdl,
              (1 << 0),
              1000,
              &IoValue,
              &Status
         );
if (result == TDRV012_OK)
{
     printf(“  Current I/O status   : 0x%08X\n”, IoValue);
     printf(“  transition status    : 0x%08X\n”, Status);
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV012_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV012_ERR_INVALID_HANDLE	The specified TDRV012_HANDLE is invalid.
TDRV012_ERR_BUSY	Too many concurrent wait jobs pending (max. 100)
TDRV012_ERR_TIMEOUT	Timeout. None of the specified events occurred.

Other returned error codes are system error conditions.

4 Legacy I/O System Functions

This chapter describes the legacy driver-level interface to the I/O system

4.1 tdrv012PciInit

NAME

tdrv012PciInit – Generic PCI device initialization

SYNOPSIS

void tdrv012PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TDRV012 device PCI spaces (base address register) and to enable the TDRV012 device for access.

The global variable *tdrv012Status* obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

Value	Meaning
> 0	Initialization successful completed. The value of tdrv012Status is equal to the number of mapped PCI spaces
0	No TDRV012 device found
< 0	Initialization failed. The value of (tdrv012Status & 0xFF) is equal to the number of mapped spaces until the error occurs. Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[]. Remedy: Add dummy entries as necessary (syslib.c).

EXAMPLE

```
extern void tdrv000PciInit();

tdrv000PciInit();
```